Schema Refinement and Normal Forms

Chapter 15

The Evils of Redundancy

- Redundancy is at the root of several problems associated with relational schemas:
  - Redundant storage/insert/delete/update anomalies
  - Integrity constraints, in particular functional dependencies, can be used to identify schemas with such problems and to suggest refinements.
  - Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD).
  - Decomposition should be used judiciously:
    - Is there reason to decompose a relation?
    - What problems (if any) does the decomposition cause?

Functional Dependencies (FDs)

- A functional dependency X→Y holds over relation R if, for every allowable instance r of R:
  - If r(1)=r(2), then X values agree, then the Y values must also agree.
  - (X and Y are sets of attributes.)
- An FD is a statement about all allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance r of R, we can check if it violates some FD if, but we cannot tell if it holds over R.

K is a candidate key for R means that K→R
  - However, K→R does not require K to be minimal!

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attilio</td>
<td>48</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-5360</td>
<td>Smerthur</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-42-5731</td>
<td>Gildu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attilio</td>
<td>48</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-5360</td>
<td>Smerthur</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-42-5731</td>
<td>Gildu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (gen, name, lot, rating, hrly_wages, hrs_worked)

  Notation: We will denote this relation schema by listing the attributes: SNLWRH
  - This is really the set of attributes (S,N,L,R,W).
  - Sometimes, we will refer to all attributes of a relation by using the relation name: (e.g., Hourly_Emps for SNLWRH)

- Some FDs on Hourly_Emps:
  - gen is the key: S→SNLWRH

  - rating determines hrly_wages: R→W

Example (Contd.)

- Problems due to R→W:
  - Update anomaly: Can we change W in just the 1st tuple of SNLWRH?
  - Insertion anomaly: What if we want to insert an employee and don’t know the hourly wage for his rating?
  - Deletion anomaly: If we delete all employees with rating 5, we lose the information about the wage for rating 5

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attilio</td>
<td>48</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smerthur</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-42-5731</td>
<td>Gildu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attilio</td>
<td>48</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smerthur</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-42-5731</td>
<td>Gildu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

Refining an ER Diagram

- 1st diagram translated:
  - Workers(S,N,L,D)
  - Departments(D,M,B)
  - Lots associated with workers,

  Suppose all workers in a dept are in the same lot: D→L

  - Redundancy fixed by:
  - Workers(S,N,L,D)
  - Dept_Lots(D,L)

  Can fine-tune this:
  - Workers(S,N,L,D)
  - Departments(D,M,B,L)
Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - A → B, B → C implies A → C

- Armstrong’s Axioms (X, Y, Z are sets of attributes):
  - Reflexivity: If X ⊆ Y, then X → Y
  - Augmentation: If X → Y, then XZ → YZ for any Z

- These are sound and complete inference rules for FDs!

Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs)
- Typically, we just want to check if a given FD X → Y is in the closure of a set of FDs F. An efficient check:
  - Compute attribute closure of X (denoted X+) wrt F:
    - Set of all attributes A such that X → A is in F+
    - There is a linear time algorithm to do this.
  - Does F = \{A B C D E\} imply A → E? (i.e., is A → E in the closure F+?) Equivalently, is E in X?

Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain normal form (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given A B: Several tuples could have the same A value, and if so, they’ll all have the same B value.

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs F is in BCNF if, for all X A in F:
  - A X (called a trivial FD), or
  - X contains a key for R.

- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

Third Normal Form (3NF)

- Reln R with FDs F is in 3NF if, for all X A in F:
  - A X (called a trivial FD), or
  - A contains a key for R.

- In other words, R is in 3NF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).
**What Does 3NF Achieve?**

- If 3NF violated by $X \rightarrow A$, one of the following holds:
  - $X$ is a subset of some key $K$
  - $X$ is a proper subset of any key.
  - There is a chain of FDs $K \rightarrow X \rightarrow A$ which means that we cannot associate an $X$ value with a $K$ value unless we also associate an $A$ value with an $X$ value.
  - Thus, 3NF is indeed a compromise relative to BCNF.

**Decomposition of a Relation Scheme**

- Suppose that relation $R$ contains attributes $A_1, \ldots, A_n$. A decomposition of $R$ consists of replacing $R$ by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of $R$ (and no attributes that do not appear in $R$), and
  - Every attribute of $R$ appears as an attribute of one of the new relations.

- Intuitively, decomposing $R$ means that we will store instances of the relations schemes produced by the decomposition, instead of instances of $R$.

- E.g., can decompose SNLW into SNLW and RW.

**Example Decomposition**

- Decompositions should be used only when needed.
  - SNLW has FDs $S \rightarrow N$ and $R \rightarrow W$.
  - Second FD causes violation of 3NF, $W$ values repeatedly associated with $R$ values.

- Easiest way to fix this is to create a relation $RW$ to store these associations, and to remove $W$ from the main schema:
  - I.e., decompose SNLW into SNLRW and RW.

- The information to be stored consists of SNLRW tuples. If we just store the projections of these tuples onto SNLRW and RW, are there any potential problems that we should be aware of?

**Problems with Decompositions**

- There are three potential problems to consider:
  - Some queries become more expensive.
  - E.g., how much did sailor Joe earn? (salary $= W$)
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation.
    - Fortunately, not in the SNLW example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
  - Fortunately, not in the SNLW example.

- **Tradeoff**: Must consider these issues vs. redundancy.

**Lossless Join Decompositions**

- Decomposition of $R$ into $X$ and $Y$ is lossless-join w.r.t. a set of FDs $F$ if, for every instance $r$ that satisfies $F$:
  - $\pi_X(r) \pi_Y(r) = r$
  - It is always true that $r \subseteq \pi_X(r) \pi_Y(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.

- Definition extended to decomposition into 3 or more relations in a straightforward way.

- It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2)).

**More on Lossless Join**

- The decomposition of $R$ into $X$ and $Y$ is lossless-join wrt $F$ if and only if the closure of $F$ contains:
  - $X \cap Y$ or $X \cap Y$
  - In particular, the decomposition of $R$ into $UV$ and $R - V$ is lossless-join if $U$ holds over $R$.
Dependency Preserving Decomposition

- Consider CS|DPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: CS|DPQV and SDP.
  - Problem: Checking JP → C requires a join.
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, Y, and on Z, then all FDs that were given to hold on R must also hold. (Claude's Problem (G1))
- Projection of set of FDs F: If R is decomposed into X, ... projection of F onto X (denoted F_X) is the set of FDs U → V in F (closure of F ) such that U, V are in X.

Dependency Preserving Decompositions (Contd.)

- Decomposition of R into X and Y is dependency preserving if (F_X union F_Y) = F^+.
  - i.e. if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
- Important to consider F^+, not F, in this definition:
  - ABC, A → B, B → C, C → A, decomposed into AB and BC.
  - Is this dependency preserving? Is C → A preserved? ????
- Dependency preserving does not imply lossless join:
  - ABC, A → B, decomposed into AB and BC.
  - And vice-versa! (Example?)

Decomposition into BCNF

- Consider relation R with FDs F. If X → Y violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF, lossless join decomposition, and guaranteed to terminate.
  - e.g.: CS|DPQV, key C, JP → C, SD → P, J → S
  - To deal with SD → P, decompose into SDP, CS|DPQV.
  - To deal with J → S, decompose CS|DPQV into JS and CS|DPQV.
- In general, several dependencies may cause violation of BCNF. The order in which we `deal with them` could lead to very different sets of relations!

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g.: CS, CS → Z, Z → C
  - Can't decompose while preserving 1st FD; not in BCNF.
- Similarly, decomposition of CS|DPQV into SDP, JS and CS|DPQV is not dependency preserving (w.r.t. the FDs JP → C, SD → P and J → S).
  - However, it is a lossless join decomposition.
  - In this case, adding JP to the collection of relations gives us a dependency preserving decomposition.
  - JP tuples stored only for checking FD! (Redundancy!)

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomposes a relation into BCNF can be used to obtain a lossless join decomposes into 3NF (typically, can stop earlier).
- To ensure dependency preservation, one idea:
  - If X → Y is not preserved, add relation XY.
  - Problem is that XY may violate 3NF! e.g., consider the addition of JP to "preserve" JP → C. What if we also have J → C?
- Refinement: Instead of the given set of FDs F, use a minimal cover for F.

Minimal Cover for a Set of FDs

- Minimal cover G for a set of FDs F:
  - G = closure of F.
  - Right hand side of each FD in G is a single attribute.
  - Intuitively, every FD in G is needed, and as small as possible in order to get the same closure as F.
- e.g. A → B, A → CD, E, EF → GH, ACDF, EG has the following minimal cover:
  - A → B, A → C, E, EF → G and EF → H
Summary of Schema Refinement

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.

- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable given typical queries), should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.